Review – CS 340 Midterm – Winter 2017 – Dr. Rodham

- UML Class Diagrams
  - Classes (name, attributes, operations)
  - Associations between classes
    - Association names
    - Role names
    - Multiplicity constraints
    - Whole/Part associations that use Aggregation or Composition
  - Generalization/Specialization
  - Notes
- Design Principles
  - Study "DesignPrinciplesReview" slides on the web site
    - Decomposition, Abstraction, Avoid Primitive Obsession, Naming, Cohesion / Single Responsibility Principle, Isolated Change Principle, Orthogonality, Minimize Dependencies, Law of Demeter, Separate Interface and Implementation, Information hiding, Algorithm & Data Structure Selection, Duplication Elimination
  - Program to abstractions (interfaces), not concretions (classes)
  - Error reporting (methods that can fail should report errors using exceptions or result objects, not just silently fail)
  - Effective encapsulation of data within a class
    - Study Phone Book example in "DataEncapsulation.txt" on the web site
    - Protect data from direct manipulation by code outside the class
    - Provide methods that clients of the class need to do their job (add, update, delete, iterate, query, etc.)
    - Create internal indexes to optimize method performance
    - Enforce data integrity, reject invalid operations
    - Provide "Can Do?" methods that clients can use to check pre-conditions, and the user interface can use to enable/disable UI components
- Design Patterns
  - Singleton
  - Command
  - Proxy
  - Façade
  - Observer
- Design by Contract
  - Method pre and post conditions
    - Who is responsible for guaranteeing pre and post conditions?
  - Class invariants
  - What happens if a method's pre-conditions are violated?

    - What does it mean if pre-conditions are met, but post-conditions are not?
    - What should a method do if pre-conditions are met, but post-conditions cannot be?
- Software Architecture
    - Definition
    - Layers
    - Dependency Inversion (program to interfaces, and the caller defines the interface through which the method call is made)
    - Model-View-Controller / Model-View-Presenter
        - Be able to explain/diagram in detail how MVC and MVP work
        - Be able to explain the difference between MVC and MVP
- Principles of Quality Assurance & Software Testing
    - Understand the ideas presented in the "IntroToTesting" slides on the web site
    - Validation & Verification (what do these words mean in the QA context?)
- Black Box Testing
    - Equivalence Partitioning. What is it? Be able to apply EP to design a set of test cases for a module.
    - Boundary Value Analysis. What is it? Be able to apply BVA to design a set of test cases for a module.
    - Other types of black box testing: Comparison testing, Performance testing, Stress testing, etc. found in "BlackBoxTesting" slides on the web site
- White Box Testing
    - Line, Branch, and Complete Condition coverage. Know what they are, how they are different from each other.
    - Loop Testing
    - Relational Condition Testing
    - Be able to apply these techniques to a piece of code to design a set of test cases for it
- Testing Strategies
    - Unit testing
        - Test-driven Development
        - What kinds of things can stubs/mocks do in a unit test scenario?
    - Integration testing
    - System Testing
        - Alpha & Beta testing
    - Regression testing. What is it? Why is it necessary? How do you design a regression test suite?